Q4]

- As per the hint, initially, all the elements are potentially maximum or minimum.
- We maintain two separate arrays for minimum and maximum
- First, we compare sets of two elements each from the main array. We add the smaller number in the array maintaining the potentially minimum numbers array while we add the greater number in the potentially maximum numbers array.
- Thus, after the previous step, we will have two arrays of n/2 length each. And the total number of comparisons made will also be **n/2. (One comparison per two elements.)**
- After we have separated the potentially max and min elements into two parts, we can evaluate minimum and maximum numbers in the array holding potentially minimum numbers and the array holding potentially maximum numbers respectively by using any comparison sorting algorithm, we know that the total number of comparisons made will be **n/2-2 for each array.**
- Thus, the total number of comparisons further to find the min and max from their respective arrays will be n-2.
- Adding n/2 from the first step and then n-2 comparisons from the consecutive steps, we get **(3n/2) – 2** comparisons.

Q5]

a) Sorting will take O(nlog(n)) time (using merge sort) and displaying the ith largest will take a constant time "i". Therefore total running time will be O(**n*log(n) + i)**

b)

a. Building a max-priority queue will take O(**nlog(n))** (heapify) time and we know that EXTRACT-MAX takes O(log(n)) time. Therefore, performing EXTRACT-MAX i times will take O(**i*log(n))** time.
b. Thus the total running time will be O(**n*log(n) + i*log(n))**
c. **Running time = O((n+i)log(n))**

c)

a. Partitioning around the ith largest number takes running time of O(n) (Via Randomized Selection Algorithm)
b. Sorting the i largest elements after the partition will take O(i*log(n)) time (using merge sort)
c. Therefore running time = **O(n + i*log(n))**